Ollscoil na Gaillimhe
University of Galway

INVESTIGATING THE KESSLER SYNDROME IN LEO USING PYTHON

Maths summer scholarship project

Author: Charles Bekaert

Supervisor: Martin Meere

c.bekaert1@universityofgalway.ie

School of Mathematical and Statistical Sciences,
University of Galway

August 2026

Contents

1 Introduction

The Kessler syndrome is a scenario proposed by NASA scientist D.J Kessler in 1978 [3]. Kessler thought of a situation where the spatial density of objects in LEO (low earth orbit) would become large enough to cause an exponential increase in collisions between objects, resulting in a possible asteroid belt.

My objective for this project is to investigate how the Kessler syndrome can arise by creating a simple python model from scratch, which models the large and small objects in orbit around the Earth, to attempt to find a relationship between the increase of objects due to collisions and time.

2 Spatial Model

This part includes the orbital and density model used for the different objects and was created using python and some packages. For the model, it was decided to split objects into two different categories, large and small objects. The large objects would be objects of a size $>$ 10cm in diameter (all objects are assumed to be spheres for simplicity), the small objects then would be $<$ 10cm in diameter. This threshold was chosen because:

1. Most satellites used by humans are $>$ 10cm in diameter and will break up into significantly more pieces than a small object.
2. There are approximately 1 million objects smaller than 10cm in diameter, which is too big of a number to simulate orbits for. [1]

Therefore the small objects were modeled using a spatial density function whereas the large objects had their orbits and trajectories calculated.

2.1 Large Objects

2.1.1 Orbital model

These satellites were modeled using orbital mechanics to get the positions of each satellite over time. The way this was done was by creating a python class named satellite, which has several attributes and methods used in the orbit propagation part of the program.

Methods

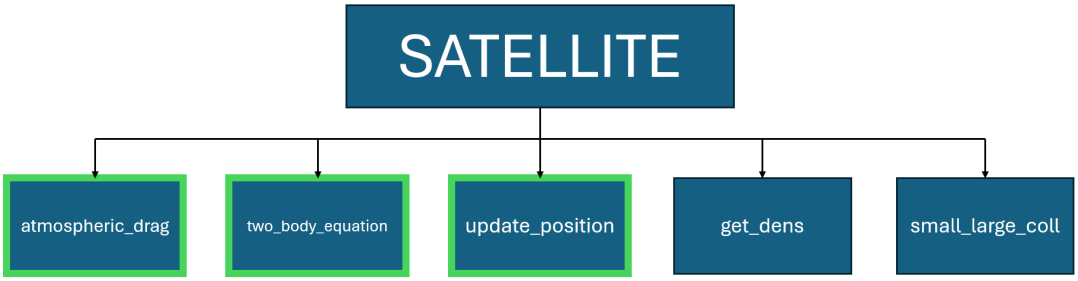

Figure 1: *Satellite class and methods, in green is part of orbit propagation*

Atmospheric drag

The first step in calculating orbit propagation for this model is calculating the atmospheric drag (uses atmospheric drag method). This is done using two different equations, the first one calculates the air density at the altitude of the object, and the second uses calculates the actual atmospheric drag force, which depends on the air density :

$$\rho = h^{-7.2} \times 10^{7.6} \tag{1}$$

where h is the altitude; this equation was obtained by using data tabulated by NASA in the Jacchia mission [4]. Using this data, a logarithmic relationship was derived between the atmospheric density of the air and the altitude:

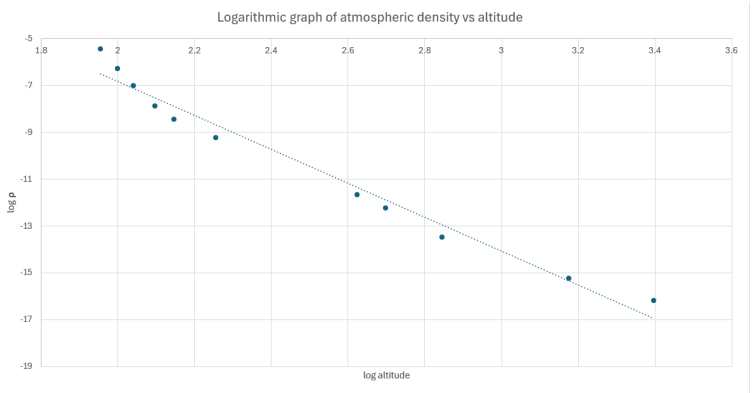

Figure 2: *Graph representing relationship between atmospheric density and altitude*

This was used since normally atmospheric air density depends on numerous complicated factors which change constantly and are not consistent throughout the entire atmosphere.

The second equation consists of the drag force calculation:

$$F_d = \frac{1}{2}\rho v^2 A C_d \tag{2}$$

Here ρ is the atmospheric density, v the velocity magnitude, A the cross sectional area and C_d the drag coefficient. For each satellite a randomized cross sectional area was given (with diameters above 10cm). The drag coefficient used was a value of 2.2, this coefficient depends heavily on the shape of the object, and numerous other complicated factors, which is why 2.2 was chosen, since it is widely recognized as a reasonable approximation [2].

The method then returns a vector value for the acceleration using the equation $F = ma$ by using F as the drag force and m as the mass of the object, and by multiplying the acceleration by the unit vector: $\frac{r}{|r|}$

Two body equations

This method simply uses Newton's law of Gravitation: $F = \frac{GMm}{r^2}$ combined with the equation of force relating to acceleration $F = ma$ to get an expression for the acceleration due to gravity:

$$a = \frac{GM}{r^2} \tag{3}$$

This acceleration value is also multiplied by the unit vector $\frac{r}{|r|}$ to get the acceleration as a vector. Then both the gravitational acceleration (3) and drag acceleration gotten by calling the atmospheric drag method are combined to get the instantaneous acceleration of the object. An array of the acceleration and velocity is returned from this method.

Update position

This method uses the scipy ode package, it turns the two body equations method into an ode function with "vode" as the integrator and "adams" as the integrator method, which proved to be the optimal choice for this program compared to other integrators. A time step of 36 seconds was used, since it needs to be long enough to try and increase performance, and isn't too long as to ignore the curvature of its path. It is important to note however that the "vode" integrator can dynamically adjust the time steps depending on the ODE.

Using this integrator with the two body equations gives an array containing the position and velocity (since the two body equations method returns array of velocity and acceleration), and so the new positions and velocities can be recorded. This update position method then continues for the number of time steps provided, and writes up these values (alongside other satellite

attributes) to npz files for later access.

2.1.2 Initialization and Simulation

As previously mentioned, each large object has its orbit modeled. The way this is done is by providing each object with randomized values for mass and cross sectional area, additionally they are all given random positions, with altitudes ranging between 200 and 2000 kilometers. Their orbits are also assumed to be approximately circular, so using the circular orbit velocity equation, the velocity is also obtained:

$$v = \sqrt{\frac{GM}{r}} \tag{4}$$

Once all of these objects have been initialized, their respective orbits can then be propagated and simulated. To increase performance, parallel processing is used to propagate many orbits at a time. These orbits are propagated for several simulation days at a time, since collisions between large objects are investigated between simulation periods. This is done since these collision probabilities are extremely low, meaning checking them every few simulated days isn't detrimental to the accuracy of the model, and takes less processing power.

Using matplotlib and its toolkits, the satellite's orbits could be visualized:

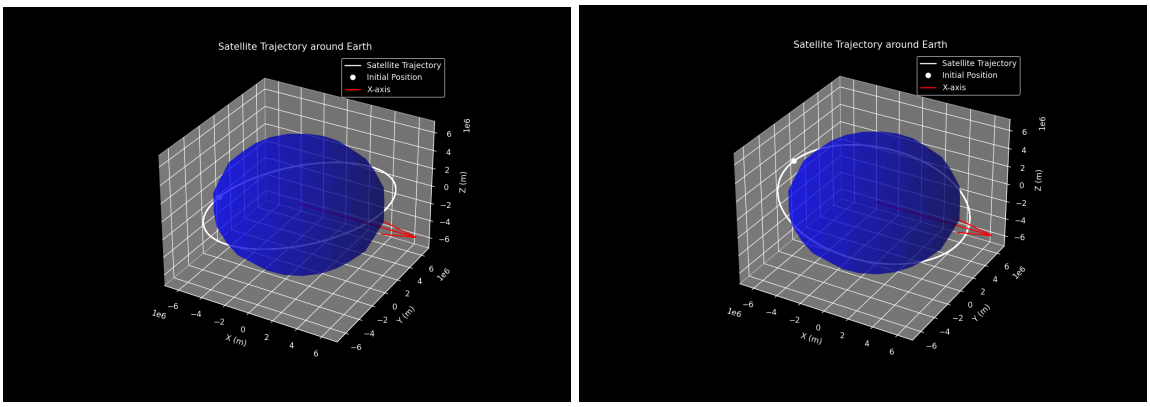

Figure 3: *Two satellites with different orbits plotted*

2.2 Small Objects

There is also a class to encapsulate all small objects:

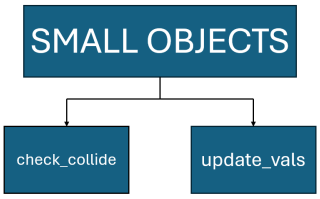

Figure 4: *Small objects class*

It contains the number of small objects for different altitudes, and thus calculates the number density of objects at this particular altitude. It is first initialized using the data gotten from ESA's 2025 spatial environment report (specifically the number of objects in certain altitude ranges) [1], then calculates the number density by dividing the number of objects by the volume of space occupied in a certain altitude range:

$$V = \int_{y_1}^{y_2} 4\pi r^2 dr \tag{5}$$

V is the volume, y_1 is the minimum altitude of the range and y_2 is the maximum altitude of the range. Uses this value of V in the equation:

$$n = \frac{N}{V} \tag{6}$$

where N is the number of objects at that altitude range, and n is the number density. This class has two methods which will be looked at in the collision model (section 3).

3 Collision Model

3.1 Collisions between two large objects

This part of the collision model reads all the large object trajectories from their respective files, and creates possible collision pairs, depending on if two large objects are in phase, and if they have similar orbits. It is separate from the satellite class. Once possible collision pairs are initialized, the program goes through each collision pair's trajectories and finds the minimum distance between the two. If the pair have a minimum distance which is below a certain threshold, determined by their relative velocities and the time steps used to calculate their trajectories: $threshold = dt \times v_{rel}$, then further calculation is done to determine whether they collide.

Since the initial time steps used for the initial trajectories are not too large (36 seconds), the two objects can be approximately treated as if they are moving in straight lines towards each

other. Their time steps are reduced to ten milliseconds, and their starting positions are located at either:

1. The position with the smallest distance between objects (for the case where the position after has the second smallest distance)
2. The position before the smallest distance between objects(for the case where this position has the second smallest distance)

If these objects' trajectories get within a distance of both of their radii combined, then they are said to collide. When this happens, they are no longer treated as large objects and are considered to have been annihilated, turning into thousands of small objects. The number of small objects produced is in a random range between 500-5000, depending on the size of the large objects that collided. This is done by creating a collision log file and appending the satellite ID, time of collision and objects produced to it.

Additionally using the small_objects class' update_vals method (fig 4), the number of small objects at that altitude is modified, and a new spatial density is calculated.

3.2 Collisions between large and small objects

This section of the collision model is included in the satellite class looked at earlier and uses these methods:

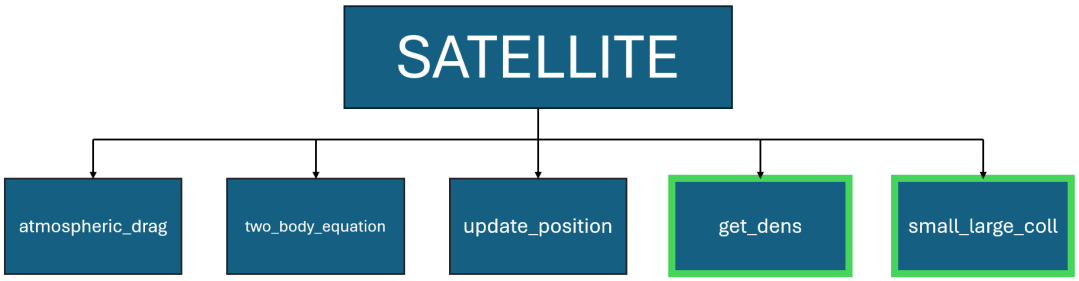

Figure 5: *Satellite class and methods, in green is part of the collision model*

For every trajectory calculation for the large objects, the program checks if a large object collides with a small one in the time frame of the time step dt. This is done in the *small_large_coll* method, it is probabilistic, and so a collision probability needs to be calculated using the spatial density of the small objects and the cross sectional area of the large ones.

Using collision theory, a collision rate between the large objects and the small objects can be calculated by using this equation:

$$I = n \times \sigma \times 7 \tag{7}$$

this is the same equation as: $I = n \times \sigma \times v$, where I is the collision rate, σ is the collisional cross sectional area and v is the velocity. The velocity in (7) is assumed to be 7km/s, since these were the average collisional velocities found in Kessler's original paper [3].

For the value of σ, it combines the radius of the large object with the small object and uses the equation for the area of a circle to get the cross sectional area. For the radius of the small object, since the only numbers which are recorded in the model are the number of objects, and not their individual sizes, an assumption must be made to get an average size for all small objects. The average is approximated to be 3cm, since there are a lot more smaller objects than large ones. [1]

Once the collision rate is obtained, the homogeneous Poisson point process formula can be utilized to find the collision probability between a large and small object:

$$P(K(t) = k) = \frac{(I\tau)^k}{k!} e^{-I\tau} \tag{8}$$

where τ is the time interval dt. Since the program is looking for the probability of an event where there is at least one collision, k is equal to one, so the equation can be rewritten in this form:

$$P = I\tau e^{-I\tau} \tag{9}$$

using python's random library, a random float is generated between 0 and 1, and if this float number is below or equal to the probability calculated, it is said that the large object has collided with a smaller one. Similarly to the process of collisions between two large objects, it is assumed that the large object here ceases to exist and breaks into hundreds of smaller objects (which is a big assumption to make). This assumption is done to reduce the complication of the program as not to decrease performance.

This is done in the same way as in the previous section, where the satellite ID is put into

the collision log (alongside the time of collision). The *small_objects* class' *update_vals* method is then called to change the number of small objects, and thus the spatial density.

3.3 Collisions between small objects

For collisions between small objects, this only involves the spatial density function. It works very similarly to the previous section, where the program checks for collisions for every trajectory calculation. One of the main differences here is that it checks for collisions using the *check_collide* method in the *small_objects* class.

For each altitude range the probability of collision between small objects is calculated (since the spatial density is already obtained). This probability is calculated by first using the equation for collision rate using spatial density:

$$I = \frac{1}{2} n^2 \sigma < v > \tag{10}$$

here, as done previously, the value of the velocity is assumed to be equal to 7km/s. For σ it is calculated by using a value of cross sectional radius of 3cm (since the total diameter is considered to be 6cm, with the assumption of each small object having a diameter/size of 3cm).

Then the same equation for the probability (9) is used to get the collision probability between small objects. The same process of using python's random module is used to determine whether a collision occurs or not, and it is assumed that a collision of this kind would fragment into a hundred or so small objects.

4 Complications and results

4.1 Improvements to be made

There were quite a few complications during this project. To start with, there were only a couple months to create a model for Earth's spatial environment alone. This, alongside the need to present and write up a report, meant many shortcuts were taken, which decreased the accuracy and performance of the program.

The first improvements that could be done is in terms of the orbital model, such as making

orbits less random. Most objects in space follow similar orbits since some altitudes and trajectories are more optimal for satellites to follow. Some examples here are equatorial and polar orbits. These are a lot more common than orbits which are in between those regions of space, and would mean the population and thus collision probability would be higher. A solution for this could be to use real data to create a distribution of satellite orbits, to still be able to randomize the orbits and have a more realistic population distribution.

Another improvement to the orbital model could be not to assume circular orbits to create more variation. Additionally, modifying the drag equation (2) and atmospheric air density equation (1) with more realistic equations, by creating an atmospheric model and not by assuming that objects are spherical, would be less approximative. Finally, a system with more than three bodies could have been considered, since the gravitational pull of the moon and the sun (alongside solar activity) play a role in the objects' trajectories.

There are also improvements to be made on the spatial density of the small objects, such as implementing atmospheric drag and introducing a decrease in objects as they re-enter the atmosphere. There is also the fact that their sizes are approximated for the collision model, and that collision consequences are approximated (this is also true between large objects).

Lastly and most importantly, the model takes a lot of processing power and is quite slow. This meant that realistic results were impossible to get (at least with the resources at my disposition), a solution to this would be to try to optimize the program further, conducting tests in smaller batches (needing a lot more time to simulate), or to use a supercomputer to obtain results. This is also the main reason why the model was simplified with many assumptions, since modeling tens of thousands of objects over a long period of time with many calculations per object is very resource expensive.

4.2 Results

The lack of resources made it impossible to conduct properly scaled simulations, meaning only tests could be conducted. When 10 000 satellites were attempted to be simulated for over 5 days at a time, the computer memory would run out. Then when checking for collisions, the collision checking model would also run out of memory. This presented a major problem, since the collision check wasn't able to run for the amount of satellites even for a small period of time. Therefore, only collision and trajectory checks were conducted, for a smaller sample of satellites.

A few different altitude ranges were used for these tests, when simulating a week, with 2000 large objects in the range of 200 to 2000km, no collisions (as expected) were detected. This is realistic, since even in a situation where there are over 15000 large objects (as is the case in 2026) [1], collisions between large objects are very rare, and only a few have been recorded.

Now when significantly decreasing the altitude range to 200.00-200.05km, over 7 days with 2000 satellites with different orbits, several collisions were detected:

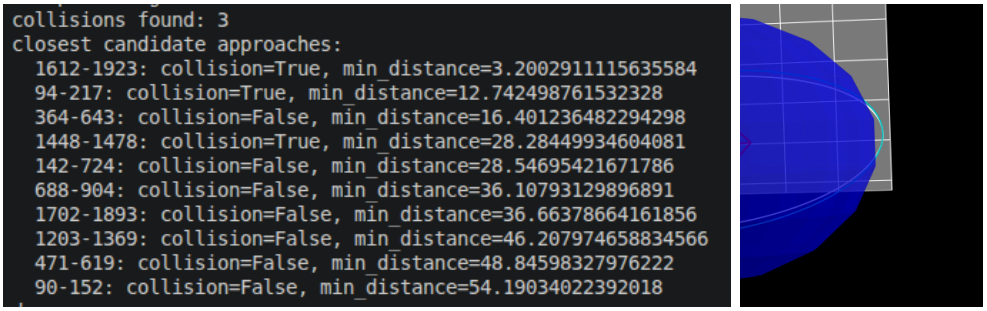

Figure 6: *Collision shown on RHS between objects 1612 and 1923*

This shows that collisions are correctly detected and calculated. It is, however, an unrealistic representation of our spatial environment. The benefit is that it presents the idea of testing/simulating within smaller ranges of space, which would require many modifications to the code and would require extra time to implement.

5 Conclusion

The model is functional and, even though many assumptions were made, large object trajectories and collisions are successfully determined. It is difficult to say, however, if the model is accurate or not due to the lengthy processing times. This means that much more time and resources are needed to get this model to work to its full potential.

From the results obtained, a development of the Kessler syndrome scenario in the near future is therefore inconclusive. This is because the sample size of the data obtained is too small to determine whether there is an exponential increase in collisions over time.

References

[1] European Space Agency. *Space Environment Report*. 2025.

[2] Peter C.E. Roberts David Mostaza Prieto Benjamin P. Graziano. *Spacecraft drag modelling*. 2013.

[3] Donald J. Kessler. *Collision Frequency of Artificial Satellites: The Creation of a Debris Belt*. 1978.

[4] NASA. *Jacchia - Lineberry Upper Atmosphere Density Model*. 1982.

Code on GitHub